# Relation Extraction

Our GoodTuring team worked on the Natural language Processing Project- "Extracting relations between two named entities in a sentence". We got support from Professor Mithun Balakrishna and our TA Takshak. Our team includes Gunjan Agicha and Shashi Shekhar.

## Problem Description

We are given a training and test dataset, where each sentence contains two entities tagged with <e1>...</e1>....<e2>...</e2> tags. We have to determine what is the relationship between these entities and what is the direction in which the relationship holds.

Eg: " <e1> Leland High School </e1> is a public high school located in the Almaden Valley in <e2> San Jose </e2> California USA in the San Jose Unified School District . "

org:city_of_headquarters(e1,e2)

In the above sentence,

Entity1 is Leland High School

Entity2 is San Jose

Relationship between the entities: org:city_of_headquarters

Direction of the relationship: (e1,e2)

Using statistical and Machine Learning techniques, our task is to extract meaningful features from the data, train the model over those extracted features and predict the correct relationship and direction between entities for the test dataset.

## Data

We are given 2 datasets for this task. Dataset1 has 17k+ training example and 3k+ testing example but has few missing entities. This dataset has relationships related to ORG, PER etc. Dataset2 is from SemEval task and has 8k training examples and 2.7k testing examples and contains relations of type Component-Whole.

## Proposed Solution

Machine Learning algorithms work on numbers and not on words directly. Thus we have to convert the given sentence into numerical features. Our proposed solution has following steps:

1. **Extract features from the sentences:**

   The features extracted from the sentence can be divided into 3 types:

   **Entity-based features:**

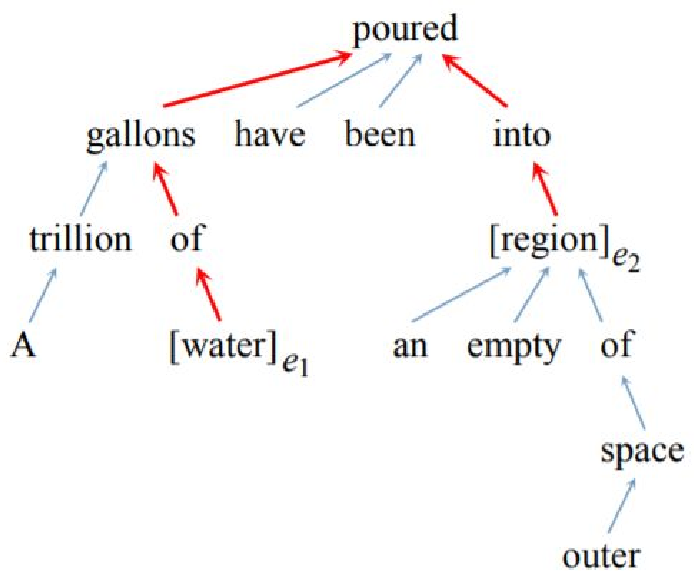
   - **Entity1**
   - **Entity2**
   - **Root of Entity1**
   - **Root of Entity2**
   - Part of Speech**(POS) E1**
   - Part of Speech**(POS) E2**
   - Name Entity Recognition**(NER) for E1**
   - Name Entity Recognition**(NER) for E2**

   **Word-based features:**

   - **Words in-between** entities
   - **Word before** E1
   - **Word after** E2

   **Syntactic features:**

   - **Constituency Parsing**: Given the entities, what type of parsing they have, NP, VP, PP etc
   - **Shortest Dependency Path(SDP)** between root E1 and root E2: Dependency chart is drawn and the words connected root words of entities are taken into consideration.



Dependency tree shown for a sentence with water and region as entities.

2. **Encode the extracted features:**

   Once we have extracted all the features, we store them into a csv file. Next step is to determine how to encode each feature.

   **Encoding POS and ENR:**

   POS and ENR are categorical features, hence we can convert them into numbers using any categorical converter, label encoding, binary encoding, one-hot encoding etc. For this use case, I have converted them using binary encoding.

   **Word Representation:**

   Many of our features are words like **Entities, root words, words in between, SDP** etc**.** The big question is how do we convert words to numbers. There are many techniques available like bag of words, TF-IDF, word2Vec, for this case we have used word2Vec from Gensim. There are pre-trained models available, but since our dataset has many name entities which are not general words, hence we have trained the model over our training and test datasets to generate encodings for our specific dataset.

   The Word2Vec model converts a word into a 100 dimensional vector. Thus a sentence with 5 words gets converted into 5X100 dimensions.

We tried two approaches for word embeddings:

1. Summed up vectors for each word within a sentence, thus each sentence is encoded into a 100 dimensional vector.
2. Found the max words in a sentence and converted each word into a 100 length vector. Calculated the difference in the max words and the current entity word length, and added zero padding equal to the difference in length*100
   Eg: If max length is 10, then encoding each word will lead to 10*100 features.

We found that encoding and summing up gave almost the same results as concatenating each word embedding. Hence, we have taken the first approach and summed up the embedding to give 100 dim vectors for entities, root words, words in between and shortest dependency path.

**3. Build a machine learning model over the training features:**

We have trained our dataset over Decision Tree, SVM and XGBoost from sklearn. We tried different combinations of features and added more features one by one. Hyper-parameter tuning was performed using GridSearchCV.

We started first by training all 3 models over encoded POS and NER tags.

Next, we trained the 3 models over entities.

Then trained over root words and similarly for rest.

We found that XGBoost gave the best performance each time. Hence when we trained on all features, we tried only the XGBoost model.

We also tried deep neural networks over a subset of features and then over all the features. The XGBoost model gave better results than the neural network.

**4. Evaluate the performance over the test dataset:**

The problem asked to calculate accuracy_score, precision_score, recall_score, f1-score all under average set as macro. There are 2 cases under which we have to calculate these metrics.

a. Both relation and direction are correct
b. Relation is correct irrespective of direction.

**Implementation Details**

On the first step of the project, we analyzed the training and test dataset. There are two entities available for every line in the dataset. Dataset also contains the relationship for every pair of entities. After analyzing the dataset properly, we decided to choose Python as a programming language to develop the solution for the given problem.

We extracted the training dataset from the training file and stored the extracted data in a data frame(of pandas). We made this dataframe the prime source of data for our solution.

Once the entire corpus is migrated from the text file to the data frame, we moved to the next step of our project, which involves processing and sanitizing the data for our project. We made sure that we select only those lines for our processing which contains both entities and have defined the relationship between them. After the sanitization of data, we worked on extracting the entities and their relationships. We stored this information as separate columns in our data frame.

In the next step, we worked on the tokenization and lemmatization of the words in the data frame using NLTK. We further made progress in our feature extraction and successfully extracted the pos tag for all the words in given sentences.

We used spacy and NLTK as a prime tool to extract the features from the dataset. We used wordnet as a source and NLTK as a tool for extracting Hypernyms, hyponyms, meronyms, and holonyms as features. We have used SpaCy as a prime tool to extract the name entity tags, part-of-speech tags, dependency parse tree and constituency parsing.

Once all the features are saved into a dataframe, we thought of ways of encoding them. As mentioned earlier, the word features were encoded using word embeddings, and categorical features were encoded using binary encoder. Each feature after encoding is stored in separate csv files, aso that we can build models on each one individually as well as build models on combined features.

Different combinations of features are tried and the metrics calculated on the 2 criteria mentioned earlier are stored into a results file. The results from the model are shown in the report later.

**Programming Tools:**

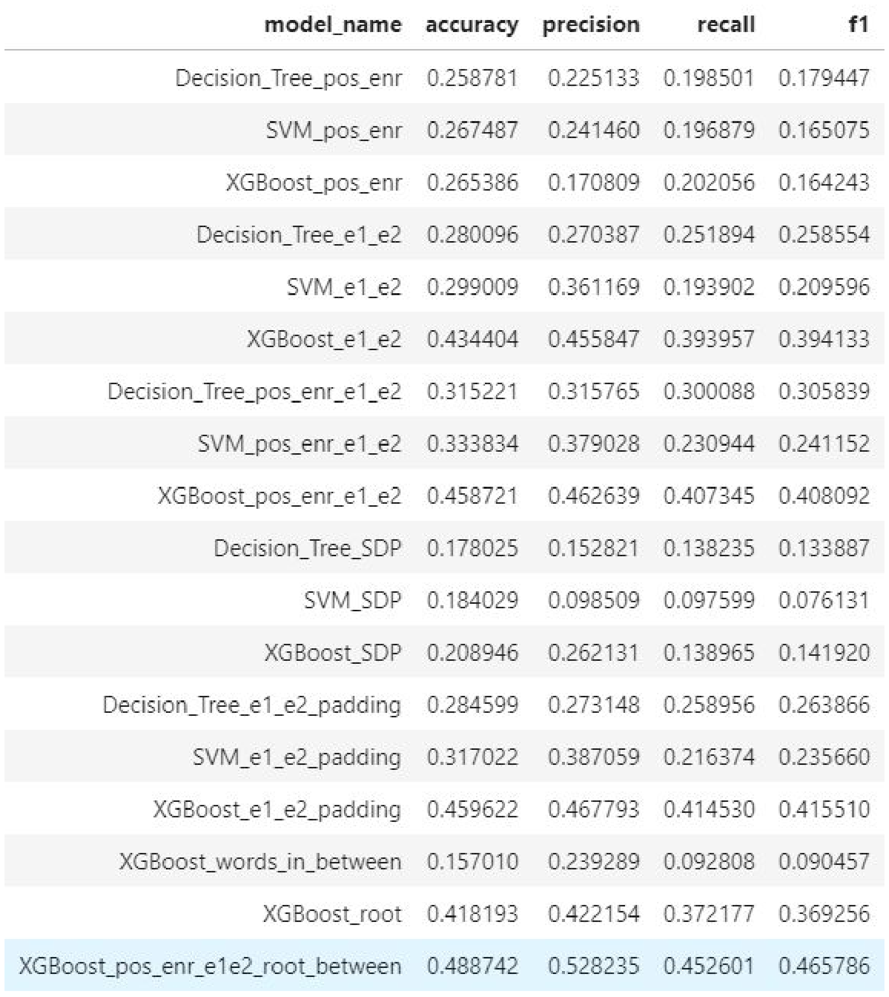
- Used Jupyter lab for coding environment and python for coding.

- To extract features from sentences, Spacy library is used. Spacy provides an easy to use API and all the functionalities for NLP like tokens, lemma, POS, NER, root, dependency tree etc.
- For model building and evaluation sklearn is used.
- For word embeddings gensim word2vec model is used.
- For binary encoding category encoders are used.
- For neural network pytorch is used.

**Results and Error Analysis**

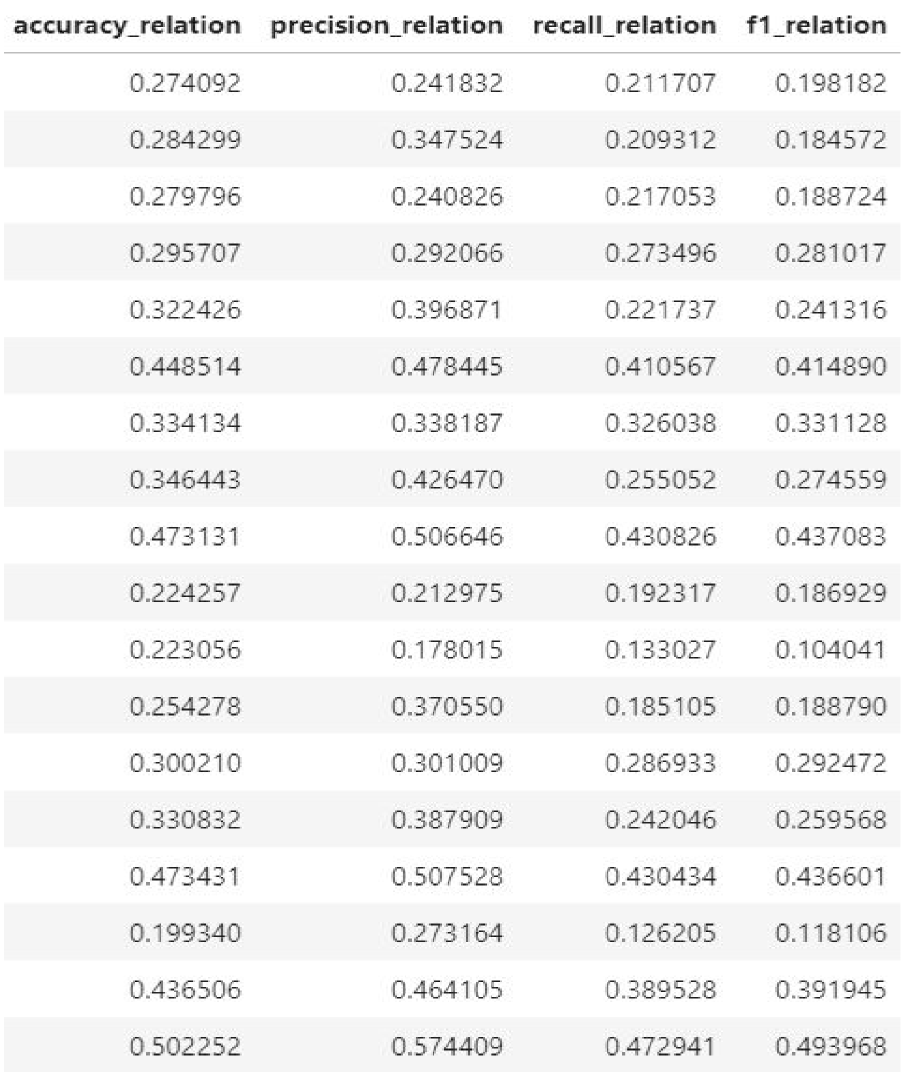
Two criteria are used for evaluation. The results from both the criteria are shown below

1. Both relation and direction are correct



| model_name | accuracy | precision | recall | f1 |
|---:|---:|---:|---:|---:|
| Decision_Tree_pos_enr | 0.258781 | 0.225133 | 0.198501 | 0.179447 |
| SVM_pos_enr | 0.267487 | 0.241460 | 0.196879 | 0.165075 |
| XGBoost_pos_enr | 0.265386 | 0.170809 | 0.202056 | 0.164243 |
| Decision_Tree_e1_e2 | 0.280096 | 0.270387 | 0.251894 | 0.258554 |
| SVM_e1_e2 | 0.299009 | 0.361169 | 0.193902 | 0.209596 |
| XGBoost_e1_e2 | 0.434404 | 0.455847 | 0.393957 | 0.394133 |
| Decision_Tree_pos_enr_e1_e2 | 0.315221 | 0.315765 | 0.300088 | 0.305839 |
| SVM_pos_enr_e1_e2 | 0.333834 | 0.379028 | 0.230944 | 0.241152 |
| XGBoost_pos_enr_e1_e2 | 0.458721 | 0.462639 | 0.407345 | 0.408092 |
| Decision_Tree_SDP | 0.178025 | 0.152821 | 0.138235 | 0.133887 |
| SVM_SDP | 0.184029 | 0.098509 | 0.097599 | 0.076131 |
| XGBoost_SDP | 0.208946 | 0.262131 | 0.138965 | 0.141920 |
| Decision_Tree_e1_e2_padding | 0.284599 | 0.273148 | 0.258956 | 0.263866 |
| SVM_e1_e2_padding | 0.317022 | 0.387059 | 0.216374 | 0.235660 |
| XGBoost_e1_e2_padding | 0.459622 | 0.467793 | 0.414530 | 0.415510 |
| XGBoost_words_in_between | 0.157010 | 0.239289 | 0.092808 | 0.090457 |
| XGBoost_root | 0.418193 | 0.422154 | 0.372177 | 0.369256 |
| XGBoost_pos_enr_e1e2_root_between | 0.488742 | 0.528235 | 0.452601 | 0.465786 |

2. Both relation and direction are correct



| accuracy_relation | precision_relation | recall_relation | f1_relation |
|---|---|---|---|
| 0.274092 | 0.241832 | 0.211707 | 0.198182 |
| 0.284299 | 0.347524 | 0.209312 | 0.184572 |
| 0.279796 | 0.240826 | 0.217053 | 0.188724 |
| 0.295707 | 0.292066 | 0.273496 | 0.281017 |
| 0.322426 | 0.396871 | 0.221737 | 0.241316 |
| 0.448514 | 0.478445 | 0.410567 | 0.414890 |
| 0.334134 | 0.338187 | 0.326038 | 0.331128 |
| 0.346443 | 0.426470 | 0.255052 | 0.274559 |
| 0.473131 | 0.506646 | 0.430826 | 0.437083 |
| 0.224257 | 0.212975 | 0.192317 | 0.186929 |
| 0.223056 | 0.178015 | 0.133027 | 0.104041 |
| 0.254278 | 0.370550 | 0.185105 | 0.188790 |
| 0.300210 | 0.301009 | 0.286933 | 0.292472 |
| 0.330832 | 0.387909 | 0.242046 | 0.259568 |
| 0.473431 | 0.507528 | 0.430434 | 0.436601 |
| 0.199340 | 0.273164 | 0.126205 | 0.118106 |
| 0.436506 | 0.464105 | 0.389528 | 0.391945 |
| 0.502252 | 0.574409 | 0.472941 | 0.493968 |

XGBoost gives the best performance when all the features are used together.

**Problems Encountered and their solutions:**

One problem we faced initially was an engineering one, how to get all the features from the list of sentences, as there are 5-6 features being extracted for each sentence, hence it was very time-taking. Also, these features have to be stored somewhere so that they are not lost causing repeated calculations.

We created a dataframe to store the sentences and relations, then applied functions to the sentences one by one and stored the results in the same dataframe using lambda functions in pandas. This one by one method helped to check if the features are created as expected and to see their results faster.

Another problem we faced was how to encode the word features as they are of variable length for each sentence. We read a few papers and found a technique to represent words in the 100 dimensional vector, word embedding format and sum up the word embeddings for each entity etc. We compared this technique v/s if we take each word and have 100 vector length for each and do 0 padding till the max length of the words. We found that both techniques gave similar results and hence stuck with summing up vectors for all the words.

**Pending Issues:**

Hypernyms, hyponyms, meronyms and holonyms all were found, but the entities are all mostly proper nouns and so their hypernyms etc do not exist. Hence it didn't make sense to have so many empty features to build the model. Thus these are not used. A better way to use these hypernyms so that they make sense and help the model has to be found.

Constituency parsing of entities are all mostly NP and models will not be learning much out of it, hence we have found them but not used to build the model.

**Potential Improvements**

Adding few features related to hypernyms, synonyms might help the model to learn better. Using Convolutional Neural Network(CNN) might help as well, as it will learn the local relationship between words and entities. Thus having smart features, and deep learning models might help build a better model.

**References:**

https://arxiv.org/pdf/1508.03720.pdf
https://github.com/roomylee/awesome-relation-extraction
https://arxiv.org/ftp/arxiv/papers/2003/2003.08615.pdf
https://arxiv.org/pdf/1903.09941.pdf
https://www.aclweb.org/anthology/N19-1298.pdf
https://www.cs.cmu.edu/~nbach/papers/A-survey-on-Relation-Extraction.pdf
http://proceedings.mlr.press/v101/zhao19a/zhao19a.pdf
https://www.aclweb.org/anthology/C18-1100.pdf
https://dl.acm.org/doi/pdf/10.3115/1219044.1219066
https://nlp.stanford.edu/mengqiu/publication/ijcnlp08.pdf
https://www.cs.utexas.edu/~ml/papers/spk-emnlp-05.pdf
https://www.aclweb.org/anthology/2020.lrec-1.135.pdf